Symmetric-key Cryptography: an Engineering Perspective

Nicky Mouha

1ESAT/COSIC, KU Leuven and iMinds, Belgium
2Project-team SECRET, Inria, France

ASK 2014 — December 19, 2014
Overview

Engineering Perspective

- Design, analysis, implementation
- Basic concepts and techniques
Overview

Engineering Perspective

- Design, analysis, implementation
- Basic concepts and techniques

Two Parts

- Hash functions
- MAC algorithms
Overview

**Engineering Perspective**
- Design, analysis, implementation
- Basic concepts and techniques

**Two Parts**
- Hash functions
- MAC algorithms

**Simplified View**
- Small inaccuracies, details missing
- Incomplete study: citations missing
Part I: Hash Functions
**Hash Function**

**Hash Function** $h$

- Generates a short “fingerprint” of a message

**Security Requirements**

- One-way function:
  given $Y$, hard to find $m : h(m) = Y$

- Collision resistant function:
  hard to find $m \neq m' : h(m) = h(m')$

- …

**SHA-3 Competition (2008-2012)**
Hash Function $h$

- Generates a short “fingerprint” of a message

Security Requirements

- One-way function:
  given $Y$, hard to find $m : h(m) = Y$

- Collision resistant function:
  hard to find $m \neq m' : h(m) = h(m')$

- ...
Permutation-Based Hash Functions

Hash Functions Based on Permutations

- Simpler to design: no key schedule
- Block-cipher-based: see later

(Cryptographic) Permutation

- Provable security: statistical object (random permutation)
- Cryptanalysis: deterministic algorithm (no “distinguishers”)

\[
\begin{array}{cccc}
  & K \downarrow & \kappa & \\
  P \rightarrow & E & C \\
  b \rightarrow & b & b
\end{array}
\]

\[
\begin{array}{cccc}
  x \rightarrow & \pi & y \\
  b \rightarrow & b & b
\end{array}
\]
Hash Function Rate $\alpha$

- $\alpha = \frac{\text{data processed per permutation call (in bits)}}{\text{permutation size (in bits)}}$

- Note: various definitions of “rate” exist!
Hash Function Rate \( \alpha \)

- \( \alpha = \frac{\text{data processed per permutation call (in bits)}}{\text{permutation size (in bits)}} \)
- Note: various definitions of “rate” exist!

Ideal Construction

- Rate-1 hash function: \( \alpha = 1 \)
Rate-1 Hash Function: First Attempt

Simplest Rate-1 Hash Function

\[ \pi(m) \]

\[ m_1 \]

\[ m_2 \]

\[ \cdots \]

\[ m_\ell \]

\[ h(m) \]
Rate-1 Hash Function: First Attempt

Collision: Correcting Block Attack

\[
m_1 \xrightarrow{n} \pi \quad m_2 \xrightarrow{\pi} \ldots \xrightarrow{x} \pi \xrightarrow{n} h(m)
\]

\[
m_1' \xrightarrow{n} \pi \quad m_2' \xrightarrow{\pi} \ldots \xrightarrow{y} \pi \xrightarrow{n} h(m)
\]
Rate-1 Hash Function: Second Attempt

Another Rate-1 Hash Function

\[
\begin{align*}
\pi(m_1) &\quad \pi(m_1, m_2) &\quad \pi(m_2) &\quad \ldots &\quad \pi(m_{\ell}) &\quad h(m) \\
\end{align*}
\]
Rate-1 Hash Function: Second Attempt

Observation

0 \rightarrow m_1 \xrightarrow{n} \pi \xrightarrow{m_1} x \xrightarrow{m_1} x \xrightarrow{0} \ldots \xrightarrow{m_\ell} \pi \xrightarrow{n} h(m)
Collision Attack (Black et al., Crypto ’02)
Impossibility Result

Black et al. (Eurocrypt ’05)
- Compression function from $n$-bit permutation
- Information-theoretic: $f_1$, $f_2$ can be any function
- Generic collision attack: at most $n + \lceil \log_2(n) \rceil$ queries
Rogaway-Steinberger (Eurocrypt ’08)

- Compression function from $k \ n$-bit permutations
- Information-theoretic: $f_i$ can be any function
- Generic collision attack: $2^{n[1-(m-0.5s)/k]}$
Rogaway-Steinberger (Eurocrypt ’08)

- Compression function from $k = 3$ $n$-bit permutations
- Information-theoretic: $f_i$ can be any function, $m = 2$, $s = 1$
- Generic collision attack: $2^{n[1-(2-0.5\cdot1)/3]} = 2^{n/2}$
Security/Efficiency Tradeoffs

Mennink-Preneel (Crypto ’12)

- Compression function from $k = 3$ $n$-bit permutations
- Constructions with only XORs, first systematic analysis
- Optimal collision resistance: $2^{n/2}$
Why Not One Big Permutation?

- $2n$-bit permutation instead of $n$-bit
- Same generic collision attack: $2^{n/2}$
- More efficient than three $n$-bit permutations?
Scaling Law

“When the input size of a symmetric-key primitive doubles, the number of operations (roughly) doubles as well”.
Scaling Law

“When the input size of a symmetric-key primitive doubles, the number of operations (roughly) doubles as well”.

Remarks

• Not intuitive: $b \rightarrow b$ bits: $(2^b)^{2^b} = 2^{b2^b}$ functions
• Not rigorous: based on design choices and attacks
• How to count “operations”?
Scaling Law

“When the input size of a symmetric-key primitive doubles, the number of operations (roughly) doubles as well”.

Remarks

- Not intuitive: $b \rightarrow b$ bits: $(2^b)^2 = 2^{b2^b}$ functions
- Not rigorous: based on design choices and attacks
- How to count “operations”?

Next Slides: Scaling Law Examples
Scaling Law: Fixed Word Size

**PHOTON: 4-bit Words**
- 100/144/196/256-bit permutation: 12 rounds
- (288-bit permutation: 12 rounds, but 8-bit word size)
Scaling Law: Fixed Word Size

**PHOTON: 4-bit Words**
- 100/144/196/256-bit permutation: 12 rounds
- (288-bit permutation: 12 rounds, but 8-bit word size)

**Rijndael (256-bit key): 8-bit Words**
- 128/192/256-bit block size: 14 rounds
Scaling Law: Fixed Word Size

PHOTON: 4-bit Words
• 100/144/196/256-bit permutation: 12 rounds
• (288-bit permutation: 12 rounds, but 8-bit word size)

Rijndael (256-bit key): 8-bit Words
• 128/192/256-bit block size: 14 rounds

Skein: 64-bit Words
• 256/512-bit block/key size: 72 rounds
• 1024-bit block/key size: 80 rounds
• Overdesign? Best (non-biclique) attack is on 36 rounds (Yu et al., SAC ’13)
Scaling Law: Variable Word Size

**BLAKE**

- 960-to-256-bit: 14 rounds (32-bit words)
- 1920-to-512-bit: 16 rounds (64-bit words)
Scaling Law: Variable Word Size

**BLAKE**
- 960-to-256-bit: 14 rounds (32-bit words)
- 1920-to-512-bit: 16 rounds (64-bit words)

**SHA-2**
- SHA-256: 768-to-256-bit: 64 rounds (32-bit words)
- SHA-512: 1536-to-512 bit: 80 rounds (64-bit words)
Scaling Law: Variable Word Size

**BLAKE**
- 960-to-256-bit: 14 rounds (32-bit words)
- 1920-to-512-bit: 16 rounds (64-bit words)

**SHA-2**
- SHA-256: 768-to-256-bit: 64 rounds (32-bit words)
- SHA-512: 1536-to-512 bit: 80 rounds (64-bit words)

**Keccak**
- 800-bit permutation: 22 rounds (32-bit words)
- 1600-bit permutation: 24 rounds (64-bit words)
- Note: zero-sum distinguisher for full-round 1600-bit permutation (Boura et al., Duan-Lai)
Scaling Law: Counterexamples?

Grøstl

- 512-bit permutation: 10 rounds
- 1024-bit permutation: 14 rounds
Scaling Law: Counterexamples?

Grøstl

- 512-bit permutation: 10 rounds
- 1024-bit permutation: 14 rounds
- Close! If 15 rounds: three small or one big: same cost
Scaling Law: Counterexamples?

Grøstl

- 512-bit permutation: 10 rounds
- 1024-bit permutation: 14 rounds
- Close! If 15 rounds: three small or one big: same cost
- Best attacks: resp. 9/10 rounds (Jean et al., FSE ’12)
Scaling Law: Counterexamples?

**Grøstl**
- 512-bit permutation: 10 rounds
- 1024-bit permutation: 14 rounds
- Close! If 15 rounds: three small or one big: same cost
- Best attacks: resp. 9/10 rounds (Jean et al., FSE ’12)

**Spongent**
- $b$-bit permutation, $r = b/2$ rounds, $b/4$ S-boxes/round: $b^2/8$ S-boxes in total
Scaling Law: Counterexamples?

**Grøstl**
- 512-bit permutation: 10 rounds
- 1024-bit permutation: 14 rounds
- Close! If 15 rounds: three small or one big: same cost
- Best attacks: resp. 9/10 rounds (Jean et al., FSE ’12)

**Sponge**
- $b$-bit permutation, $r = b/2$ rounds, $b/4$ S-boxes/round: $b^2/8$ S-boxes in total
- Four $n$-bit or one $2n$-bit permutation: same cost
Scaling Law: Counterexamples?

**Grøstl**
- 512-bit permutation: 10 rounds
- 1024-bit permutation: 14 rounds
- Close! If 15 rounds: three small or one big: same cost
- Best attacks: resp. 9/10 rounds (Jean et al., FSE ’12)

**Spongent**
- \( b \)-bit permutation, \( r = b/2 \) rounds, \( b/4 \) S-boxes/round: \( b^2/8 \) S-boxes in total
- Four \( n \)-bit or one \( 2n \)-bit permutation: same cost
- 272-bit Spongent: 5x lower throughput than 256-bit PHOTON (Bogdanov et al., IEEE Trans. Comp. 2013)
Hash Functions with $2^{n/2}$ Collision Resistance

**Rate-1 Hash Function ($\alpha = 1$)**

- Impossible (Black et al., Eurocrypt ’05)
- Generic collision attack: at most $n + \lceil \log_2(n) \rceil$
Hash Functions with $2^{n/2}$ Collision Resistance

**Rate-1 Hash Function ($\alpha = 1$)**

- Impossible (Black et al., Eurocrypt ’05)
- Generic collision attack: at most $n + \lceil \log_2(n) \rceil$

**Rate-0.5 Hash Function ($\alpha = 0.5$)**

- Three $n$-bit permutations
- One $2n$-bit permutation
Hash Functions with $2^{n/2}$ Collision Resistance

Rate-1 Hash Function ($\alpha = 1$)
- Impossible (Black et al., Eurocrypt ’05)
- Generic collision attack: at most $n + \lceil \log_2(n) \rceil$

Rate-0.5 Hash Function ($\alpha = 0.5$)
- Three $n$-bit permutations
- One $2n$-bit permutation

Higher Rate Possible? ($0.5 < \alpha < 1$)
- Yes, arbitrarily close to $\alpha = 1$!
- See next slide...
Sponge Function

\[ \alpha = \frac{r}{r+c} \]

Example

- SHA3-256: \( c = 512, r + c = 1600, \alpha = 0.68 \)
Concatenate-Permute-Truncate

\[ \alpha = \frac{r}{r+c} \]

Example

- Grindahl-256: \( r = 32, \ r + c = 416, \ \alpha = 0.08 \)
  (Note: low \( \alpha \), but compensated by weak \( \pi \))
Merkle-Damgård with Davies-Meyer

- $\alpha = \frac{r}{r+c}$

Example

- SHA256: $c = 256$, $r = 512$, $\alpha = 0.67$
Considerations

**Lightweight**
- Small hardware implementation
- Achieved by small permutation!
- Typically very low $\alpha$
Considerations

Lightweight

- Small hardware implementation
- Achieved by small permutation!
- Typically very low $\alpha$

Simplicity

- e.g. JH: one 1024-bit permutation for all output sizes
- Downside: not best tradeoff for small outputs
Considerations

**Lightweight**
- Small hardware implementation
- Achieved by small permutation!
- Typically very low $\alpha$

**Simplicity**
- e.g. JH: one 1024-bit permutation for all output sizes
- Downside: not best tradeoff for small outputs

**Other Criteria**
- Software: register pressure, instruction set, parallelism,…
- Hardware: throughput, latency, power, energy,…
- Both: message length, reuse of function/library, secure implementation, interoperability, standards compliance,…
Conclusion

Permutation-Based Hash Functions

- Engineering approach
- Tradeoffs for theory/cryptanalysis/implementations
- Simplified model: inaccuracies in figures, designs
Conclusion

Permutation-Based Hash Functions

- Engineering approach
- Tradeoffs for theory/cryptanalysis/implementation
- Simplified model: inaccuracies in figures, designs

Goal

- Help to understand design choices
- No intention to criticize certain designs!
- Feedback is welcome
Part II: MAC Algorithms
Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers

Nicky Mouha\textsuperscript{1}, Bart Mennink\textsuperscript{1}, Anthony Van Herrewege\textsuperscript{1}, Dai Watanabe\textsuperscript{2}, Bart Preneel\textsuperscript{1}, Ingrid Verbauwhede\textsuperscript{1}

\textsuperscript{1}ESAT/COSIC, KU Leuven and iMinds, Belgium
\textsuperscript{2}Yokohama Research Laboratory, Hitachi, Japan

Presented at SAC 2014
MAC Algorithm for Microcontrollers

Message Authentication Code (MAC)

- \( \text{MAC}_K(m) = \tau \)
- Authenticity, no confidentiality
- Same key for MAC generation and verification
MAC Algorithm for Microcontrollers

Message Authentication Code (MAC)

- $\text{MAC}_K(m) = \tau$
- Authenticity, no confidentiality
- Same key for MAC generation and verification

Microcontroller

- Cheap 8/16/32-bit processor: USD 25-50¢
- Applications: home, medical, industrial,...
- Ubiquitous: 30-100 in any recent car
Design

Requirements

• Drop-in replacement for AES-CMAC (variant of CBC-MAC for variable-length messages)
• Same functionality and security
Design

Requirements

- Drop-in replacement for AES-CMAC (variant of CBC-MAC for variable-length messages)
- Same functionality and security

Speed

- “Ten times faster than AES”
Design

Requirements

- Drop-in replacement for AES-CMAC (variant of CBC-MAC for variable-length messages)
- Same functionality and security

Speed

- “Ten times faster than AES”

Approach

- Dedicated design for microcontrollers
Commonly used MACs

Based on (cryptographic) hash function

- **Example**: HMAC, SHA3-MAC
- Large block size, collision resistance unnecessary
Commonly used MACs

Based on (cryptographic) hash function

- **Example:** HMAC, SHA3-MAC
- Large block size, collision resistance unnecessary

Based on universal hashing

- **Examples:** UMAC, GMAC, Poly1305
- **Requires:** nonce, constant-time multiply, long tags
Commonly used MACs

Based on (cryptographic) hash function

- **Example**: HMAC, SHA3-MAC
- Large block size, collision resistance unnecessary

Based on universal hashing

- **Examples**: UMAC, GMAC, Poly1305
- **Requires**: nonce, constant-time multiply, long tags

Based on block cipher

- **Example**: CMAC
Commonly used MACs

Based on (cryptographic) hash function
- **Example**: HMAC, SHA3-MAC
- Large block size, collision resistance unnecessary

Based on universal hashing
- **Examples**: UMAC, GMAC, Poly1305
- **Requires**: nonce, constant-time multiply, long tags

Based on block cipher
- **Example**: CMAC
- **Problem**: ten times too slow!
Our Approach

Every cycle counts!

- Avoid load/store: keep data in registers
- Avoid bit masking
- Make optimal use of instruction set
Our Approach

**Every cycle counts!**
- Avoid load/store: keep data in registers
- Avoid bit masking
- Make optimal use of instruction set

**Bridging the gap**
- Cryptanalysis
- Provable security
- Implementation
Which primitive?

- Cryptographic hash function \( \times \)
Primitive

Which primitive?
- Cryptographic hash function \(\times\)
- Universal hash function \(\times\)
Primitive

Which primitive?

- Cryptographic hash function ✗
- Universal hash function ✗
- Block cipher ✗
Primitive

Which primitive?

• Cryptographic hash function ✗
• Universal hash function ✗
• Block cipher ✗
• Ideal permutation ✗
Which primitive?

- Cryptographic hash function $\times$
- Universal hash function $\times$
- Block cipher $\times$
- Ideal permutation $\times$

$\rightarrow$ Even-Mansour Block Cipher $\checkmark$
Primitive

Which primitive?

- Cryptographic hash function \( \times \)
- Universal hash function \( \times \)
- Block cipher \( \times \)
- Ideal permutation \( \times \)

\( \rightarrow \) Even-Mansour Block Cipher \( \checkmark \)

Related-key attacks

- Insecure: choose uniformly random keys!
Chaskey: Mode of Operation

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$
- $K_1 = 2K$
Chaskey: Mode of Operation

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$
Chaskey: Mode of Operation: Phantom XORs

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$
Chaskey: Mode of Operation: Phantom XORs

- Split \( m \) into \( \ell \) blocks of \( n \) bits
- Top: \( |m_\ell| = n \), bottom: \( 0 \leq |m_\ell| < n \)
- \( K_1 = 2K, K_2 = 4K \)
Chaskey: Mode of Operation: Block-cipher-based

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$
Chaskey: Mode of Operation: Block-cipher-based

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$

variant of FCBC [BR’00]
Chaskey: Mode of Operation: Compared to CMAC

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$

variant of CMAC [IK’03]
Chaskey: Mode of Operation: Compared to CMAC

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$

$\begin{array}{c}
E_K(0^n) \rightarrow K
\end{array}$

variant of CMAC [IK'03]
Chaskey: Mode of Operation: Compared to CMAC

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$

$E_K(0^n) \rightarrow K$

variant of CMAC [IK’03]

1. Chaskey Mode
   - $m_1$ $m_2$ ... $m_\ell$
   - $E_K \| K$ $E_K \| K$ ... $E_K \| K$
   - $K \oplus K_1$
   - $\tau$

2. Even-Mansour
   - $m_1$ $m_2$ ... $m_\ell$
   - $E_K \| K$ $E_K \| K$ ... $E_K \| K$
   - $K \oplus K_2$
   - $\tau$
Chaskey: Mode of Operation: Compared to CMAC

- Split $m$ into $\ell$ blocks of $n$ bits
- Top: $|m_\ell| = n$, bottom: $0 \leq |m_\ell| < n$
- $K_1 = 2K$, $K_2 = 4K$

1. $E_K(0^n) \rightarrow K$

variant of CMAC [IK’03]

3. Not in CMAC

Even-Mansour

\[
E_K \oplus K_1
\]

\[
E_K \oplus K_2
\]
Cryptanalysis

**MAC forgery:** find new valid $(m, \tau)$
- $D$: data complexity (\# chosen plaintexts)
- $T$: time complexity (\# permutation eval.)

**Attacks**
- Internal collision: $D \approx 2^{n/2}$
- Key recovery: $T \approx 2^n / D$
- Tag guessing: $\approx 2^t$ guesses

**Chaskey parameters**
- Key size, block size: $n = 128$, tag length: $t \geq 64$
Permutation

Design

- Add-Rot-XOR (ARX)
- Inspired by SipHash
- 32-bit words
- 8 rounds

Properties

- Rotations by 8, 16: faster on 8-bit µC
- Fixed point: $0 \rightarrow 0$
- Cryptanalysis: rotational, (truncated) differential, MitM, slide,... see paper!
**Chaskey: Speed Optimized (gcc -O2)**

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Algorithm</th>
<th>Data [byte]</th>
<th>ROM [byte]</th>
<th>Speed [cycles/byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-M0</td>
<td>AES-128-CMAC</td>
<td>16</td>
<td>13492</td>
<td>173.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>13492</td>
<td>136.5</td>
</tr>
<tr>
<td></td>
<td>Chaskey</td>
<td>16</td>
<td>1308</td>
<td>21.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>1308</td>
<td>18.3</td>
</tr>
<tr>
<td>Cortex-M4</td>
<td>AES-128-CMAC</td>
<td>16</td>
<td>28524</td>
<td>118.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>28524</td>
<td>105.0</td>
</tr>
<tr>
<td></td>
<td>Chaskey</td>
<td>16</td>
<td>908</td>
<td>10.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>908</td>
<td>7.0</td>
</tr>
</tbody>
</table>
## Chaskey: Size Optimized (gcc -Os)

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Algorithm</th>
<th>Data [byte]</th>
<th>ROM [byte]</th>
<th>Speed [cycles/byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-M0</td>
<td>AES-128-CMAC</td>
<td>16</td>
<td>11664</td>
<td>176.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>11664</td>
<td>140.0</td>
</tr>
<tr>
<td></td>
<td>Chaskey</td>
<td>16</td>
<td>414</td>
<td>21.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>414</td>
<td>16.9</td>
</tr>
<tr>
<td>Cortex-M4</td>
<td>AES-128-CMAC</td>
<td>16</td>
<td>10925</td>
<td>127.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>10925</td>
<td>89.4</td>
</tr>
<tr>
<td></td>
<td>Chaskey</td>
<td>16</td>
<td>402</td>
<td>16.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>402</td>
<td>11.2</td>
</tr>
</tbody>
</table>
Conclusion and Current Status

**Chaskey:**
MAC algorithm for 32-bit microcontrollers

- Addition-Rotation-XOR (ARX)
- Even-Mansour block cipher
- ARM Cortex-M: \(7-15 \times\) faster than AES-128-CMAC
Conclusion and Current Status

**Chaskey:**
MAC algorithm for 32-bit microcontrollers

- Addition-Rotation-XOR (ARX)
- Even-Mansour block cipher
- ARM Cortex-M: 7-15× faster than AES-128-CMAC

**Standardization**

- Chaskey: currently in study period
- ISO/IEC JTC1 SC27: MAC standardization
- ITU-T SG17: crypto for IoT, ITS
Questions?
Supporting Slides
Security Proof

**MAC forgery:** find new valid \((m, \tau)\)

- \(D\): block cipher (PRP) queries
- \(T\): permutation queries

### Standard Model

\[
\text{Adv}^\text{mac}_{\text{Chaskey-B}}(q, D, r) \leq \frac{2D^2}{2^n} + \frac{1}{2^t} + \text{Adv}^3_{E}(D, r)
\]

### Ideal Permutation Model

\[
\text{Adv}^\text{mac}_{\text{Chaskey}}(q, D, r) \leq \frac{2D^2}{2^n} + \frac{1}{2^t} + \frac{D^2 + 2DT}{2^n}
\]